RECEIVED
CENTRAL FAX CENTER

JAN 0 6 2006

Yee & Associates, P.C.

4100 Alpha Road
Suite 1100
Dallas, Texas 75244

Main No. (972) 385-8777
Facsimile (972) 385-7766

Facsimile Cover Sheet

To: Commissioner for Patents for **Examiner John J. Romano** **Group Art Unit 2192**	Facsimile No.: 571/273-8300
From: Hope Cichra for Lourdes Perez Legal Assistant to Peter B. Manzo	No. of Pages Including Cover Sheet: 29

Message:

Enclosed herewith:

- Transmittal Document; and
- Appeal Brief.

Re: Docket No: AUS920010996US1 Serial No. 10/042,079

Date: Friday, January 06, 2006

Please contact us at (972) 385-8777 if you do not receive all pages indicated above or experience any difficulty in receiving this facsimile.	*This Facsimile is intended only for the use of the addressee and, if the addressee is a client or their agent, contains privileged and confidential information. If you are not the intended recipient of this facsimile, you have received this facsimile inadvertently and in error. Any review, dissemination, distribution, or copying is strictly prohibited. If you received this facsimile in error, please notify us by telephone and return the facsimile to us immediately.*

PLEASE CONFIRM RECEIPT OF THIS TRANSMISSION BY FAXING A CONFIRMATION TO 972-385-7766.

RECEIVED
CENTRAL FAX CENTER
JAN 0 6 2006

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Broussard** §
Serial No.: **10/042,079** §
Filed: **January 7, 2002** §
For: **Data Processing System, Method, and Computer Program Product for Generating a Generic Compilation Interface from Object-Oriented Code** §

Group Art Unit: **2192**
Examiner: **Romano, John J.**
Attorney Docket No.: **AUS920010996US1**

35525
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

Certificate of Transmission Under 37 C.F.R. § 1.8(a)
I hereby certify this correspondence is being transmitted via facsimile to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, facsimile number (571) 273-8300 on January 06, 2006.

By: [illegible]
Hope Cichra

TRANSMITTAL DOCUMENT

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:
ENCLOSED HEREWITH:

- Appeal Brief (37 C.F.R. 41.37)

A fee of $500.00 is required for filing an Appeal Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0447. No additional fees are believed to be necessary. If, however, any fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0447.

Respectfully submitted,

Peter B. Manzo
Registration No. 54,700

Duke W. Yee
Registration No. 34,285
YEE & ASSOCIATES, P.C.
P.O. Box 802333
Dallas, Texas 75380
(972) 385-8777
ATTORNEYS FOR APPLICANT

RECEIVED
CENTRAL FAX CENTER

JAN 0 6 2006

Docket No. AUS920010996US1

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Broussard**	§	
	§	Group Art Unit: **2192**
Serial No. **10/042,079**	§	
	§	Examiner: **Romano, John J.**
Filed: **January 7, 2002**	§	
	§	
For: **Data Processing System, Method, and Computer Program Product for Generating a Generic Compilation Interface from Object-Oriented Code**	§ § § §	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Certificate of Transmission Under 37 C.F.R. § 1.8(a)
I hereby certify this correspondence is being transmitted via facsimile to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, facsimile number (571) 273-8300 on January 06, 2006.

By: [illegible]
Hope Ochra

01/09/2006 TL0111 00000044 090447 10042079
01 FC:1402 500.00 DA

APPEAL BRIEF (37 C.F.R. 41.37)

This brief is in furtherance of the Notice of Appeal, filed in this case on November 9, 2005.

The fees required under § 41.20(B)(2), and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

REAL PARTY IN INTEREST

The real party in interest in this appeal is the following party:

International Business Machines Corporation of Armonk, New York.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-39.

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: none.
2. Claims withdrawn from consideration but not canceled: none.
3. Claims pending: 1-39.
4. Claims allowed: none.
5. Claims rejected: 1-39.
6. Claims objected to: none.

C. CLAIMS ON APPEAL

The claims on appeal are: 1-39.

STATUS OF AMENDMENTS

An amendment after Final Rejection was not filed. Therefore, claims 1-39 on appeal herein are as amended in the Response to Office Action dated June 14, 2005.

SUMMARY OF CLAIMED SUBJECT MATTER

A. CLAIM 1 - INDEPENDENT

The subject matter of claim 1 is directed to a method in a data processing system (Figures 1-3 and associated text on page 9, line 29 – page 15, line 3) for generating a generic compilation interface (page 6, lines 8-10), from a first object-oriented software package (page 6, lines 10-12). All public classes included in the first object-oriented software package are identified (page 6, lines 12-14). All of the public entities included in each of the public classes are identified (page 6, lines 15-17). All references to software that is defined in a second object-oriented software package are removed from the public entities included in each of the public classes (page 6, lines 25-27). An equivalent public class for each of the identified public classes is generated and the equivalent public class includes equivalent public entities that include no references to the software defined in the second object-oriented software package (page 6, lines 27-31). Each of the equivalent public classes are compiled (page 7, lines 5-7) and a compilation interface for the first object-oriented software package includes each of the compiled equivalent public classes (page 7, lines 7-10).

B. CLAIM 14 – INDEPENDENT

Independent method claim 1 is representative of independent apparatus claim 14. As a result, the claimed subject matter of independent claim 14 is found in the same locations as claim 1 as laid out above. In addition, the "means for" performing each of the features of claim 14 may be found in Figures 1-3 and the associated text on page 9, line 29 – page 15, line 3.

C. CLAIM 27 – INDEPENDENT

Independent method claim 1 is representative of independent computer program product claim 27. As a result, the claimed subject matter of independent claim 27 is found in the same locations as claim 1 as laid out above. In addition, the "instruction means" for performing each of the features of claim 27 may be found on page 17, lines 10-13.

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. GROUND OF REJECTION 1 (Claims 1, 2, 4-7, 9-15, 17-20, 22-28, 30-33, and 35-39)

Claims 1, 2, 4-7, 9-15, 17-20, 22-28, 30-33, and 35-39 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Krishna et al., U.S. Patent Publication No. 2003/0051233 A1 in view of Dale Green, *Trail: The Reflection API*, The Java Tutorial (Nov. 27, 1999) <http://java.sun.com/docs/books/tutorial/reflect/>.

B. GROUND OF REJECTION 2 (Dependent Claims 3, 16, and 29)

Dependent claims 3, 16, and 29 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Krishna et al., U.S. Patent Publication No. 2003/0051233 A1 in view of Dale Green, *Trail: The Reflection API*, The Java Tutorial (Nov. 27, 1999) <http://java.sun.com/docs/books/tutorial/reflect/> and further in view of Evans et al., U.S. Patent No. 6,836,884 B1.

C. GROUND OF REJECTION 3 (Dependent Claims 8, 21, and 34)

Dependent claims 8, 21, and 34 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Krishna et al., U.S. Patent Publication No. 2003/0051233 A1 in view of Dale Green, *Trail: The Reflection API*, The Java Tutorial (Nov. 27, 1999) <http://java.sun.com/docs/books/tutorial/reflect/> and further in view of obviousness.

ARGUMENT

A. GROUND OF REJECTION 1 (Claims 1, 2, 4-7, 9-15, 17-20, 22-28, 30-33, and 35-39)

The Examiner rejected claims 1, 2, 4-7, 9-15, 17-20, 22-28, 30-33, and 35-39 under 35 U.S.C. § 103(a) as being unpatentable over Krishna et al., U.S. Patent Publication No. 2003/0051233 A1 ("Krishna") in view of Dale Green, *Trail: The Reflection API*, The Java Tutorial (Nov. 27, 1999) <http://java.sun.com/docs/books/tutorial/reflect/> ("Green"). This rejection is respectfully traversed.

The Examiner bears the burden of establishing a *prima facie* case of obviousness based on the prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). For an invention to be *prima facie* obvious, the prior art must teach or suggest all claim limitations. *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974). In this case, the Examiner has not met this burden because all of the features of these claims are not found in the cited references as believed by the Examiner. Therefore, the combination of Krishna and Green would not reach the presently claimed invention recited in these claims.

Independent method claim 1 of the present invention, which is representative of independent apparatus claim 14 and independent computer program product claim 27, reads as follows:

> 1. A method in a data processing system for generating a generic compilation interface, from a first object-oriented software package, said method comprising the steps of:
> identifying all public classes included in said first object-oriented software package;
> for each of said public classes, identifying all public entities included in each of said public classes;
> removing all references to software that is defined in a second object-oriented software package from said public entities included in each of said public classes;
> generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second object-oriented software package;
> compiling each of said equivalent public classes; and
> generating a compilation interface for said first object-oriented software package including each of said compiled equivalent public classes.

With regard to claim 1, the Examiner states:

In regard to claim 1, **Krishna** discloses:

- *"A method in a data processing system for generating a generic compilation interface from a first object-oriented software package, said method comprising the steps of..."* (E.g., see Figure 7A & Page 2, Paragraph [0025]), wherein the library stubs exclude the source code executable statements, but include (generic), declarations and interfaces so that the secondary developer can compile class files for converting to CAP files, etc (interface).
- *"...removing all references to software that is defined in a second software package from said public entities included in each of said public classes..."* (E.g., see Figure 7B & Page 2, Paragraph [0025]), wherein the library stubs exclude (remove), the source code executable statements (software defined in a second package), but include declarations and interfaces so that the secondary developer can compile class files, wherein Figure 7B, steps 721-734, teach replacing a reference returned with the appropriate return type value.
- *"... generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second package..."* (E.g., see Figure 7A & 7B, steps 737-747 & Page 3, Paragraph [0036]), wherein equivalent public class for each of said identified public class are generated, wherein "... <u>only non-private (public) method signatures and field signatures are needed for off-card compiling and conversion.., is sufficient for synthesizing the library stubs 220 (Figure 3)</u>".
- "*...compiling each of said equivalent public classes; and generating a compilation interface for said first package including each of said compiled equivalent public classes.*" (E.g., see Figure 7A & 7B & Page 3, Paragraph [0048]), wherein the pseudo code teaches generating (compiling) an equivalent JAR file (Step 747).

But **Krishna** does not expressly disclose "*...identifying all public classes included in said first software package ...*" or "*..for each of said public classes, identifying all public entities included in each of said public classes ...*". However, **Green** discloses:

- "*... identifying all public classes included in said first software package .*"(E.g., see "Discovering Class Modifiers", Page 7), wherein all public class modifiers are discovered.

> - "*... for each of said public classes, identifying all public entities included in each of said public classes* ..." (E.g., see "Trail: The Reflection API", Page 1, Paragraph 1), wherein all public class modifiers are discovered along with their fields, methods and variables (entities).
>
> **Krishna** and **Green** are analogous art because they are both concerned with the same field of endeavor, namely, using the JAVA language to examine, manipulate and work with classes. Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine a public class modifiers and their attributes with **Krishna's** JAVA program for interpreting, interfacing and compiling. The motivation to do so, is suggested by **Krishna**, "... only non-private method signatures and field signatures are needed... for compiling...", (Page 3, Paragraph [0035]). Furthermore, **Green** suggests "...to use the reflection API if you are writing development tools..." (Page 1, Paragraph 1).

Final Office Action dated August 26, 2005, pages 11-13.

Krishna teaches a method for Java™ Card application development. Krishna, page 1, paragraph 0019. A first developer writes Java™ Card library source code and compiles the library into a package of Java™ Card library classes. Krishna, page 1, paragraph 0019. "If a second developer wants to create an applet that uses services of the first developer's applet, the first developer provides the source code version of the first applet to the second developer. One issue with this process concerns protection of the first developer's intellectual property. That is, secrecy of the first developer's source code is jeopardized by distributing it to secondary developers." Krishna, page 1, paragraph 0006. Hence, the method as taught by Krishna is concerned with protecting the first developer's intellectual property rights in source code that the first developer wrote for the Java™ Card library.

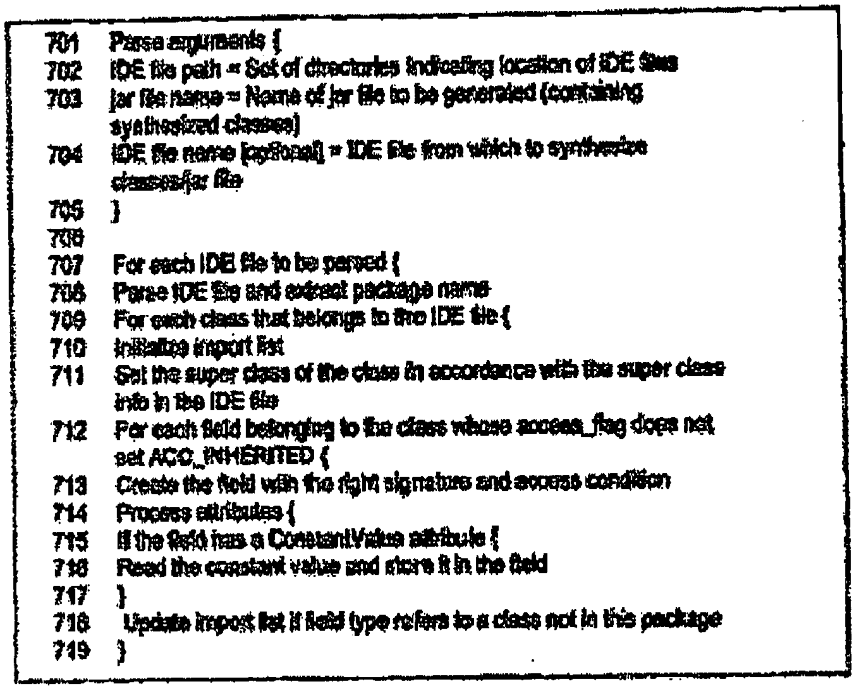
In addition, Krishna teaches that "Java™ Cards do not support dynamic loading of classes. Therefore, in order to be suitable for interpretation by the Java™ Virtual Machine, a Java™ Card executable <u>must include</u> all elements necessary for on-card interpretation, such as <u>cross-references to code *outside the executable itself*</u>." Krishna, page 1, paragraph 0019. [Emphasis added]. In other words, Krishna teaches that the cross-referenced source code is outside the executable source code written by the first developer and must be included within the Java™ Card application in order to be suitable for interpretation by the Java™ Virtual Machine.

In contrast, the present invention recites in claim 1 generating a generic compilation interface by removing all references to software that is defined in a second object-oriented software package. Thus, the present invention removes all references, or cross-references, to any other software applications within the generated compilation interface as recited in claim 1. Krishna instead teaches that cross-references to other software applications must be included in the Java™ Card application. Consequently, Krishna does not teach or suggest this claim 1 feature.

However, the Examiner cites Krishna, page 2, paragraph 0025, as teaching removing all references to software that is defined in a second object-oriented software package recited in claim 1. This Examiner-cited passage states that since distribution of the Java™ Card library source code poses an unacceptable risk to the first developer's intellectual property rights in the source code written by the first developer, a set of library stubs are derived manually by the first developer from the Java™ Card library source code and provided to the second developer as part of distributed information. The library stubs exclude the Java™ Card library source code executable statements, but includes declarations and interfaces of the source code so that the secondary developer can compile class files. Krishna, page 2, paragraph 0025.

The source code executable statements mentioned in the passage from Krishna above refers to source code written by the first developer that the first developer has intellectual property rights to. In other words, Krishna teaches that the first developer manually excludes all source code written by the first developer from the library stub to protect intellectual property rights, while leaving all other statements, including declarations and interfaces, within the library stub for the second developer to build upon. As shown above, Krishna teaches that the cross-referenced source code is considered outside the executable statements written by the first developer and is not removed or excluded from the library stub. In fact, Krishna teaches that all cross-references must be included within the Java™ Card application in order for it to function within the Java™ Virtual Machine, which provides the environment for the Java™ Card application to perform its tasks. Krishna makes no reference to removing all references to software that is defined in a second object-oriented software package as is recited in claim 1. Further, Krishna does not suggest the desirability of, or the motivation for, removing all other software applications source code references.

In addition, removing all references to software that is defined in a second object-oriented software package to generate a generic compilation interface as recited in claim 1 is distinguishable from manually removing executable source code to protect intellectual property rights of the first developer as taught by Krishna. Further, Krishna teaches updating the import list if the field type refers to a class not in the package. Krishna, Figure 7A, line 718, below.



```
701  Parse arguments {
702  IDE file path = Set of directories indicating location of IDE files
703  jar file name = Name of jar file to be generated (containing
     synthesized classes)
704  IDE file name [optional] = IDE file from which to synthesize
     classes/jar file
705  }
706
707  For each IDE file to be parsed {
708  Parse IDE file and extract package name
709  For each class that belongs to the IDE file {
710  Initialize import list
711  Set the super class of the class in accordance with the super class
     info in the IDE file
712  For each field belonging to the class whose access_flag does not
     set ACC_INHERITED {
713  Create the field with the right signature and access condition
714  Process attributes {
715  If the field has a ConstantValue attribute {
716  Read the constant value and store it in the field
717  }
718   Update import list if field type refers to a class not in this package
719  }
```

FIG. 7A

In other words, Krishna teaches updating the integrated development environment's (IDE) import list to include references to a class of objects not contained within the executable source code package. The IDE, or export file, which contains non-intellectual property distributed information from the first developer, is sent to the second developer. Krishna, page 2, paragraph 0030 and Figure 3. As shown above, this non-intellectual property distributed information contained in the IDE must include cross-references to other source code "outside the executable itself" in order for the Java™ Virtual Machine to interpret the Java™ Card application properly as taught by Krishna, whereas, the present invention removes all references to software that is defined in a second object-oriented software package as recited in claim 1.

Furthermore, as shown in Figure 7B below, Krishna teaches that the stub generator processes any method that is not overridden by creating a method header with the right signature and access condition, setting the method name to correspond to the class name if the method is a constructor method, setting the return type as the return value if the method returns a reference, and creating information from an Exceptions attribute if the method has an Exceptions attribute. Krishna, page 3, paragraph 0048 – page 4, paragraph 0049.

```
720  For each method belonging to the class whose access_flag does not set
     ACC_INHERITED {
721  Create the method header with the right signature and access condition
722  If method name begins with <init> (a constructor method) {
723  Set name to correspond to the class name
724  Set return type to null
725  }
726  If the method returns a reference
727  Set the return value to null
728  Else
729  Set the return value to 0 cast appropriately to match the method's return type
730  Process attributes {
731  If the method has an Exceptions attribute {
732  Read the fields attribute and Store the throws clause(s)
733  }
734  }
735  Update import list if method return or argument type refers to class not
     in this package
736  }
737  Synthesize class file {
738  Create a source file named <classname>.java in the appropriate directory
739  Append package statement
740  Append the imports list
741  Append the class/interface statement with superclass/superinterface list
742  Append field declarations
743  Append method stubs
744  Compile the generated source
745  }
746  }
747  Place the synthesized class files in a Jar file
748  }
```

FIG. 7B

In lines 726-729 of Figure 7B above, Krishna determines if the method returns a reference. If so, the return value is set to return a null object. Otherwise, the return value is set to 0 and the return value is cast to match the method's return type. Thus, if the return type of the method is an integer, the return value of 0 is cast to an integer. However, the return value is merely a return object, which is either a null object or an integer object. The return value is not a reference to software defined in a second software object-oriented software package.

Moreover, Krishna teaches in line 735 of Figure 7B, that the import list is updated to refer to classes that are not in the software package if the method return or if the argument type refers to classes that are not in the software package, whereas claim 1 recites removing references to software defined in a second software. Thus, instead of removing the reference to software that is outside the software package, Krishna updates the import list to include a reference to software that is outside the package in the import list. Therefore, Krishna does not teach or suggest removing all references to software that is defined in a second object-oriented software package from the public entities included in each of the public classes as recited in claim 1 of the present invention.

Green fails to cure the deficiencies of Krishna. With regard to the Green prior art reference, the Examiner states:

> In regard to Applicants' argument that *Green* does not teach or suggest removing all references to software that is defined in a second object-oriented software package (Page 18 of 23, last paragraph), it is noted that *Green* is not relied upon in the previous office action to teach removing all references to software that is defined in a second object-oriented software package.

Final Office Action dated August 8, 2005, page 8.

Green teaches a reflection application programming interface (API) that represents "classes, interfaces, and objects in the current Java™ Virtual Machine." Green, page 1, paragraph 1. By using the API, a user may "determine the class of an object, get information about the class's modifiers, fields, methods, constructors, and superclasses, find out what constants and method declarations belong to the interface, create an instance of a class whose name is unknown until runtime, get and set the value of an object's field even if the field name is unknown until runtime, invoke a method on an object that is unknown until runtime and create a new array, whose size and component type are not known until runtime and modify the array's components." Green, page 1, paragraph 1.

While Green provides the capability of invoking a method on an object that is unknown until runtime, Green does not teach or suggest removing all references to software that is defined in a second object-oriented software package as recited in claim 1. As Green merely invokes a method of a class, Green does not remove references that refer to a second object-oriented software package from the class. Therefore, Green also does not teach or suggest this recited claim 1 feature of the present invention.

Because Krishna and Green do not teach or suggest removing all references to software that is defined in a second object-oriented software package as recited in independent claims 1, 14, and 27, the combination of Krishna and Green cannot teach or suggest this recited feature.

esent invention. In particular, Krishna and
es to software that is defined in a second
pendent claims 1, 14, and 27. This feature

en teach or suggest removing all references to
software package as recited in independent
Green, and obviousness cannot teach or
21, and 34 of the current invention also are
on allowable claims. Accordingly, the
able over Krishna in view of Green and further

Peter B. Manzo

Peter B. Manzo
Reg. No. 54,700
YEE & ASSOCIATES, P.C.
PO Box 802333
Dallas, TX 75380
(972) 385-8777

Thus, Krishna and Green do not teach or suggest all features recited in claims 1, 14, and 27 of the present invention. Accordingly, the rejection of independent claims 1, 14, and 27 as being unpatentable over Krishna in view of Green has been overcome.

In view of the arguments above, independent claims 1, 14, and 27 are in condition for allowance. Claims 2, 4-7, 9-13, 15, 17-20, 22-26, 28, 30-33, and 35-39 are dependent claims depending on independent claims 1, 14, and 27, respectively. Consequently, claims 2, 4-7, 9-13, 15, 17-20, 22-26, 28, 30-33, and 35-39 also are allowable, at least by virtue of their dependence on allowable claims.

B. GROUND OF REJECTION 2 (Dependent Claims 3, 16, and 29)

The Examiner rejected dependent claims 3, 16, and 29 under 35 U.S.C. § 103(a) as being unpatentable over Krishna in view of Green and further in view of Evans et al., U.S. Patent No. 6,836,884 B1 ("Evans"). This rejection is respectfully traversed.

As shown in Section A above, neither Krishna nor Green teach or suggest all the features recited in independent claims 1, 14, and 27 of the present invention. In particular, Krishna and Green do not teach or suggest removing all references to software that is defined in a second object-oriented software package as recited in independent claims 1, 14, and 27. This feature also is not taught or suggested by Evans.

Therefore, because Krishna, Green, and Evans do not teach or suggest removing all references to software that is defined in a second object-oriented software package as recited in independent claims 1, 14, and 27, the combination of Krishna, Green, and Evans cannot teach or suggest this recited feature. As a result, dependent claims 3, 16, and 29 of the current invention also are allowable at least by virtue of their dependence upon allowable claims. Accordingly, the rejection of claims 3, 16, and 29 as being unpatentable over Krishna in view of Green and further in view of Evans has been overcome.

C. GROUND OF REJECTION 3 (Dependent Claims 8, 21, and 34)

The Examiner rejected dependent claims 8, 21, and 34 under 35 U.S.C. § 103(a) as being unpatentable over Krishna in view of Green and further in view of obviousness. This rejection is respectfully traversed.

As shown in Section A above, neither Krishna nor Green teach or suggest all the features recited in independent claims 1, 14, and 27 of the present invention. In particular, Krishna and Green do not teach or suggest removing all references to software that is defined in a second object-oriented software package as recited in independent claims 1, 14, and 27. This feature also is not obvious.

Therefore, because neither Krishna nor Green teach or suggest removing all references to software that is defined in a second object-oriented software package as recited in independent claims 1, 14, and 27, the combination of Krishna, Green, and obviousness cannot teach or suggest this recited feature. As a result, claims 8, 21, and 34 of the current invention also are allowable at least by virtue of their dependence upon allowable claims. Accordingly, the rejection of claims 8, 21, and 34 as being unpatentable over Krishna in view of Green and further in view of obviousness has been overcome.

Peter B. Manzo

Peter B. Manzo
Reg. No. 54,700
YEE & ASSOCIATES, P.C.
PO Box 802333
Dallas, TX 75380
(972) 385-8777

CLAIMS APPENDIX

The text of the claims involved in the appeal are:

1. A method in a data processing system for generating a generic compilation interface, from a first object-oriented software package, said method comprising the steps of:

identifying all public classes included in said first object-oriented software package;

for each of said public classes, identifying all public entities included in each of said public classes;

removing all references to software that is defined in a second object-oriented software package from said public entities included in each of said public classes;

generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second object-oriented software package;

compiling each of said equivalent public classes; and

generating a compilation interface for said first object-oriented software package including each of said compiled equivalent public classes.

2. The method according to claim 1, wherein said step of identifying all public entities included in each of said public classes further comprises the step of identifying all entities included in each of said public classes that include a public modifier.

3. The method according to claim 1, further comprising the steps of:

determining whether each of said entities includes a native attribute;

in response to a determination that each of said entities includes a native attribute, removing said native attribute from each of said entities; and

generating equivalent entities that include no native attributes.

4. The method according to claim 1, wherein the step of identifying all public entities included in each of said public classes further comprises the step of identifying all public methods included in each of said public classes.

5. The method according to claim 1, wherein the step of identifying all public entities included in each of said public classes further comprises the step of identifying all public parameters included in each of said public classes.

6. The method according to claim 1, wherein the step of identifying all public entities included in each of said public classes further comprises the step of identifying all public fields included in each of said public classes.

7. The method according to claim 1, wherein said step of identifying all public classes included in said first object-oriented software package further comprises the step of identifying all public classes included in a Java Archive file.

8. The method according to claim 1, wherein the step of identifying all public classes included in said first object-oriented software package further comprises the step of identifying all public classes included in said first object-oriented software package utilizing a java.util.jar utility.

9. The method according to claim 1, wherein the step of for each of said public classes, identifying all public entities included in each of said public classes further comprises the step of for each of said public classes, identifying all public entities included in each of said public classes utilizing Java Reflection.

10. The method according to claim 1, wherein said step of generating an equivalent public class for each of said identified public classes further comprises the step of generating a separate .java file for each of said identified public classes.

11. The method according to claim 10, wherein said step of compiling each of said equivalent public classes further comprises the step of compiling each said .java file.

12. The method according to claim 11, wherein said step of generating a compilation interface for said first object-oriented software package including each of said compiled equivalent public classes further comprises the steps of:

generating a compilation Java Archive file; and

storing each said compiled .java file in said compilation Java Archive file.

13. The method according to claim 1, further comprising the step of utilizing said compilation interface within an Integrated Development Environment.

14. A data processing system for generating a generic compilation interface from a first object-oriented software package, said system comprising:

means for identifying all public classes included in said first object-oriented software package;

means for each of said public classes, for identifying all public entities included in each of said public classes;

means for removing all references to software defined in a second object-oriented software package from said public entities included in each of said public classes;

means for generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second object-oriented software package;

means for compiling each of said equivalent public classes; and

means for generating a compilation interface for said first object-oriented software package including each of said compiled equivalent public classes.

15. The system according to claim 14, wherein said means for identifying all public entities included in each of said public classes further comprises means for identifying all entities included in each of said public classes that include a public modifier.

16. The system according to claim 14, further comprising:

means for determining whether each of said entities includes a native attribute;

in response to a determination that each of said entities includes a native attribute, means for removing said native attribute from each of said entities; and

means for generating equivalent entities that include no native attributes.

17. The system according to claim 14, wherein said means for identifying all public entities included in each of said public classes further comprises means for identifying all public methods included in each of said public classes.

18. The system according to claim 14, wherein said means for identifying all public entities included in each of said public classes further comprises means for identifying all public parameters included in each of said public classes.

19. The system according to claim 14, wherein said means for identifying all public entities included in each of said public classes further comprises means for identifying all public fields included in each of said public classes.

20. The system according to claim 14, wherein said means for identifying all public classes included in said first object-oriented software package further comprises means for identifying all public classes included in a Java Archive file.

21. The system according to claim 14, wherein said means for identifying all public classes included in said first object-oriented software package further comprises means for identifying all public classes included in said first object-oriented software package utilizing a java.util.jar utility.

22. The system according to claim 14, wherein said means for each of said public classes, for identifying all public entities included in each of said public classes further comprises means for each of said public classes, for identifying all public entities included in each of said public classes utilizing Java Reflection.

23. The system according to claim 14, wherein said means for generating an equivalent public class for each of said identified public classes further comprises means for generating a separate .java file for each of said identified public classes.

24. The system according to claim 23, wherein said means for compiling each of said equivalent public classes further comprises means for compiling each said .java file.

25. The system according to claim 24, wherein said means for generating a compilation interface for said first object-oriented software package including each of said compiled equivalent public classes further comprises:

means for generating a compilation Java Archive file; and

means for storing each said compiled .java file in said compilation Java Archive file.

26. The system according to claim 14, further comprising means for utilizing said compilation interface within an Integrated Development Environment.

27. A computer program product in a data processing system for generating a generic compilation interface from a first object-oriented software package, said computer program product comprising:

instruction means for identifying all public classes included in said first object-oriented software package;

instruction means for each of said public classes, for identifying all public entities included in each of said public classes;

instruction means for removing all references to software defined in a second software object-oriented software package from said public entities included in each of said public classes;

instruction means for generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second object-oriented software package;

instruction means for compiling each of said equivalent public classes; and

instruction means for generating a compilation interface for said first object-oriented software package including each of said compiled equivalent public classes.

28. The product according to claim 27, wherein said instruction means for identifying all public entities included in each of said public classes further comprises instruction means for identifying all entities included in each of said public classes that include a public modifier.

29. The product according to claim 27, further comprising:

instruction means for determining whether each of said entities includes a native attribute;

instruction means in response to a determination that each of said entities includes a native attribute, for removing said native attribute from each of said entities; and

instruction means for generating equivalent entities that include no native attributes.

30. The product according to claim 27, wherein said instruction means for identifying all public entities included in each of said public classes further comprises instruction means for identifying all public methods included in each of said public classes.

31. The product according to claim 27, wherein said instruction means for identifying all public entities included in each of said public classes further comprises instruction means for identifying all public parameters included in each of said public classes.

32. The product according to claim 27, wherein said instruction means for identifying all public entities included in each of said public classes further comprises instruction means for identifying all public fields included in each of said public classes.

33. The product according to claim 27, wherein said instruction means for identifying all public classes included in said first object-oriented software package further comprises instruction means for identifying all public classes included in a Java Archive file.

34. The product according to claim 27, wherein said instruction means for identifying all public classes included in said first object-oriented software package further comprises instruction means for identifying all public classes included in said first object-oriented software package utilizing a java.util.jar utility.

35. The product according to claim 27, wherein said instruction means for each of said public classes, for identifying all public entities included in each of said public classes further comprises instruction means for each of said public classes, for identifying all public entities included in each of said public classes utilizing Java Reflection.

36. The product according to claim 27, wherein said instruction means for generating an equivalent public class for each of said identified public classes further comprises instruction means for generating a separate .java file for each of said identified public classes.

37. The product according to claim 36, wherein said instruction means for compiling each of said equivalent public classes further comprises instruction means for compiling each said .java file.

38. The product according to claim 37, wherein said instruction means for generating a compilation interface for said first object-oriented software package including each of said compiled equivalent public classes further comprises:

instruction means for generating a compilation Java Archive file; and

instruction means for storing each said compiled .java file in said compilation Java Archive file.

39. The product according to claim 27, further comprising instruction means for utilizing said compilation interface within an Integrated Development Environment.

EVIDENCE APPENDIX

There is no evidence to be presented.

RELATED PROCEEDINGS APPENDIX

There are no related proceedings.